

Algorithms for Scheduling Weighted Packets with Deadlines in a Bounded Queue

Fei Li*

February 7, 2009

Abstract

Motivated by the Quality-of-Service (QoS) buffer management problem, we consider online scheduling of packets with hard deadlines in a finite capacity queue. At any time, a queue can store at most $b \in \mathbb{Z}^+$ packets. Packets arrive over time. Each packet is associated with a non-negative value and an integer deadline. In each time step, only one packet is allowed to be sent. Our objective is to maximize the total value gained by the packets sent by their deadlines in an online manner. Due to the Internet traffic's chaotic characteristics, no stochastic assumptions are made on the packet input sequences. This model is called a *finite-queue model*.

We use competitive analysis to measure an online algorithm's performance versus an unrealizable optimal offline algorithm who constructs the worst possible input based on the knowledge of the online algorithm. For the finite-queue model, we first present a deterministic 3-competitive memoryless online algorithm. Then, we give a randomized $(\phi^2 = ((1 + \sqrt{5})/2)^2 \approx 2.618)$ -competitive memoryless online algorithm.

The algorithmic framework and its theoretical analysis include several interesting features. First, our algorithms use (possibly) modified characteristics of packets; these characteristics may not be same as those specified in the input sequence. Second, our analysis method is different from the classical potential function approach. We use a simple charging scheme, which depends on a clever modification (during the course of the algorithm) on the packets in the queue of the optimal offline algorithm. We then prove that a set of invariants holds at the end of each time step. Finally, we analyze the two proposed algorithm in a relaxed model, in which packets have no hard deadlines but an order. We conclude that both algorithms have the same competitive ratios in the relaxed model.

1 Introduction

In the last three decades, routers in the Internet continue supporting more and more applications. Currently, most routers forward packets in a First-In-First-Out (FIFO) manner and treat all packets equally. However, the diversity of applications has resulted in heterogeneity and unpredictable network traffic. Thus, it is more reasonable to consider differentiation among packets from different types of applications (see [23, 1, 17, 2] and the references therein). For instance, we could specify values for packets to represent their priorities. Also, we may like to assign hard deadlines to packets in time-critical applications. These concerns have made buffer management at routers significant in providing effective quality of service (QoS) to various applications.

*Department of Computer Science, George Mason University. lifeli@cs.gmu.edu. Part of this work appears in the Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM 2009) [19].

One kind theoretical research on QoS buffer management starts from three paper by Aiello et al. [1], by Kesselman et al. [17] and by Hajak [15], where a model called a *bounded-delay model* is proposed. In this model, time is discrete. Packets arrive over time, and they are buffered upon arrivals. The queue capacity is unlimited. An arriving packet p has a non-negative *value* $w_p \in \mathbb{R}^+$ and an integer *deadline* $d_p \in \mathbb{Z}^+$ by which it should be transmitted; after d_p , p expires. In each time step, at most one packet can be sent. The objective is to maximize the *weighted throughput*, which is defined as the total value of the transmitted packets by their deadlines. Fig. 1 illustrates the functionalities of the online buffer management algorithms, which process newly arriving packets and send one packet out of the buffer in each time step. The buffer size and deadlines of packets limit the number of *pending packets*¹ in the queue.

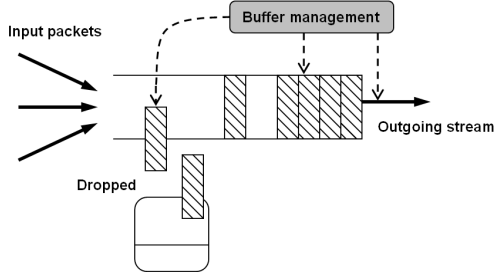


Figure 1: Buffer management is in charge of processing arriving packets from the input streams and delivering packets out of the buffer as outgoing streams.

Realizing that the capacity of a queue buffering packets is limited and such queue is a shared resource for multiplexing packets inside routers, Azar and Levy extend the single buffer bounded-delay model to multiple buffers. They consider scheduling packets with deadlines in multiple finite capacity buffers, all of which have the same finite capacities [5]. In this paper, we study the single queue scheduling problem, in which the capacity of the queue is finite.

In the ideal case, if the release time, value, and deadline of each packet are known ahead of time, an optimal schedule can be found efficiently; we call this the *optimal offline algorithm*. For instance, given no constraint over the queue capacity, the the optimal schedule can be found by computing a maximum weighted matching on a convex bipartite graph. However, we do not know all such information ahead of time. Rather, packets arrive *online*, and we only learn about a packet and its associated characteristics when it actually arrives. Furthermore, without a realistic model of the network traffic [23, 14], we cannot achieve good stochastic performance guarantee. Hence, we study the worst-case analysis for algorithms in queueing packets, without any assumptions over the input sequence.

Competitiveness has been widely accepted as the metric to measure an online algorithm's worst-case performance in theoretical computer science [9]. In this paper, we design and analyze better deterministic and randomized online algorithms, in terms of competitive ratio, for scheduling packets in a finite capacity queue. A deterministic (randomized) online algorithm ON_d (ON_r) is called *k-competitive* if its (expected) weighted throughput on *any* instance is at least $1/k$ of the weighted throughput of an

¹A pending packet is a packet in the buffer whose deadline has not expired yet. For a given time step, every pending packet is eligible for sending.

optimal offline algorithm on this instance:

$$k = \max_{\mathcal{I}} \frac{\text{OPT}(\mathcal{I}) - \alpha}{\text{ON}_d(\mathcal{I})}, \quad \text{ON}_d \text{ is a deterministic algorithm}$$

$$k = \max_{\mathcal{I}} \frac{\text{OPT}(\mathcal{I}) - \alpha}{\mathbf{E}[\text{ON}_r(\mathcal{I}, r)]}, \quad \text{ON}_r \text{ is a randomized algorithm}$$

where α is a constant, $\text{OPT}(\mathcal{I})$ is the optimal solution of an input \mathcal{I} , and r is the set of random variables flipped by a randomized online algorithm ON_r . The parameter k is known as the online algorithm's *competitive ratio* [9]². If k is not a constant, such an algorithm with competitive ratio k is called *non-competitive*. If the additive constant $\alpha \leq 0$, the algorithm ON_d (ON_r) is called *strictly k -competitive*. Note that for a randomized algorithm, the role of randomization is purely internal to the randomized algorithm. No stochastic assumption is made on the input. Also note that the optimal offline algorithm is called the *adversary* of the online algorithm since the input sequence constructed by the offline optimal algorithm is allowed to maximize the competitive ratio k .

1.1 Problem setting. The model we study here is called the *finite queue model*, in which we consider scheduling packets with deadlines in a finite capacity queue. In our model, time is discrete. Packets arrive over time and each packet p is associated with a non-negative weight $w_p \in \mathbb{R}^+$ and an integer deadline $d_p \in \mathbb{Z}^+$. d_p specifies the time by which p should be sent. If p is transmitted by its deadline d_p , p contributes our objective by a value w_p . (We use “value” and “weight”, “queue” and “buffer” interchangeably.) This model is preemptive, which means that packets already existing in the queues can be dropped at any time before they are served. If a packet is dropped, this packet cannot be delivered any more.

There is only one queue with a limited capacity of $b \in \mathbb{Z}^+$. At any time, the queue can store no more than b packets. Packets that expire are dropped immediately. At most one packet can be sent in each time step. Our target is to maximize the total value of the packets sent by their deadlines. Remember that in a bounded delay model [17, 15], the queue capacity is infinite. The finite queue model generalizes the bounded delay model: if the queue capacity is larger than any packets *slack time*, which is defined as the difference between the packets deadline and its release time, the finite queue model is the bounded-delay model.

1.2 Related work. Since the first QoS buffer management model (the bounded-delay model) was introduced in [1, 2], many researchers have considered this model as well as its variants [17, 15, 11, 10, 12, 20, 21, 13]. Most of these studies consider the single queue case, which does not have an explicit limit over the queue capacity. The best known lower bound of competitive ratio of deterministic algorithms is $\phi = (1 + \sqrt{5})/2 \approx 1.618$ [15, 11, 4]; this lower-bound also applies to instances in which the deadlines of the packets (weakly) increase with their release dates.

For an arbitrary deadline instance, a simple greedy algorithm that always schedules a maximum-value packet in the queue is 2-competitive [15, 17]. A generalization of the greedy algorithm, called EDF_α , which schedules the earliest packet with a value at least $1/\alpha$ ($\alpha \geq 1$) of the maximum-value of a packet [10], has a competitive ratio of (asymptotically) 2. Chrobak et al. [12] discuss a clever

²In real-time scheduling terminologies, $1/k$, the reciprocal of the competitive ratio, is called *competitive factor*. We use the term *competitive ratio*, which is widely recognized in the area of online computation.

modification that results in an algorithm with a competitive ratio of $64/33 \approx 1.939$. This algorithm employs a status bit to help schedule packets and hence, it is not memoryless³.

For instances in which the deadlines of the packets (weakly) increase with their release dates, Li et al. [20] propose an optimal deterministic online algorithm **MG** whose competitive ratio is ϕ . Unfortunately, this improved competitive ratio is achieved by exploiting the deadline assumption; on general instances, **MG**, like the greedy algorithm and EDF_α , is also 2-competitive. Applying similar analysis approach, but in a more complicated way, Li et al. provide a $3/\phi \approx 1.854$ -competitive deterministic algorithm [20] for the general model. Independently, Englert and Westermann present a 1.894-competitive deterministic memoryless algorithm and a $(2\sqrt{2} - 1 \approx 1.828)$ -competitive deterministic algorithm [13], which is not memoryless. Closing the gap of $[1.618, 1.828]$ between the lower bounds and the upper bounds of competitive ratio of deterministic online algorithms is still a difficult open problem. Randomization on the bounded-delay model is considered in [10]. A randomized online algorithm with a competitive ratio of $e/(e - 1) \approx 1.582$ is proposed. The lower bound of competitive ratio of randomized algorithms is 1.25. How to tighten the gap of $[1.25, 1.582]$ in the randomized model still remains open.

Azar and Levy consider the multi-buffer model in which multiple queues are bounded in their capacities and packets can have arbitrary deadlines. Notice that if the queues are unlimited in capacities, this model is the same as the bounded-delay single queue model. The lower bound 1.618 for the bounded-delay model directly applies on the multi-queue model. In [5], the authors give a deterministic memoryless 9.82-competitive algorithm. In this paper, we improve the lower bound to 2 for a family of deterministic online algorithms (this lower bound also applies to the finite queue model). To our knowledge, there is no published work of randomized algorithms on the multi-buffer model. Also, our proposed single finite queue model, which is more realistic for buffer management, has not been addressed in recent literature. For the single queue case, the competitive ratio of the algorithm in [5] 9.82 still applies.

There has also been work on another model in which the queue capacity is bounded. In this model, packets have no deadlines but weights, and the **FIFO** discipline is enforced in delivering packets [22, 18, 8] — packets should be sent in the same order as they arrive. Some researchers also consider packet scheduling in multiple **FIFO** input queues connecting one output queue [6, 7, 3, 16]: Every queue obeys the **FIFO** constraint in delivering weighted packets and each arriving packet has only one destined queue.

1.3 Our contributions. This paper provides theoretical bounds for algorithms on the finite queue model, which considers the finite capacity constraint for buffer management under a more practical modeling. Our main contributions include

1. A strictly 3-competitive deterministic memoryless online algorithm **ME** for the finite queue model (in Section 2).
2. A strictly $(\phi^2 = ((1 + \sqrt{5})/2)^2 \approx 2.618)$ -competitive randomized memoryless online algorithm **RME** for the finite queue model (in Section 3).
3. A new analysis method including a charging scheme and a set of invariants.

Table 1 summaries the competitive ratios of those known algorithms for the bounded-delay model, its variants, and our results on the finite queue model.

³An algorithm is called *memoryless* if this algorithm makes its scheduling decision only based on the packets in the current queue but not on the historical information.

Models	Upper bound of competitive ratio	Lower bound of competitive ratio
General bounded-delay model	Deterministic algorithms: 1.854 [21] 1.828 [13] Randomized algorithms: $e/(e-1) \approx 1.582$ [10]	Deterministic algorithms: 1.618 [17, 15] Randomized algorithms: 1.25 [10]
Agreeable deadline bounded-delay model	Deterministic algorithms: 1.618 [20]	Deterministic algorithms: 1.618 [17, 15]
General finite queue model	Deterministic algorithms: 3 (in this paper) Randomized algorithms: 2.618 (in this paper)	A broad family of deterministic algorithms: 2 (in this paper) -

Table 1: Summary of competitive ratios for the bounded-delay model and the finite queue model. Upper bounds are achieved by some known algorithms. Any online algorithm cannot achieve a competitive ratio less than the lower bound.

To supplement our work on the finite queue model, we also provide an optimal offline algorithm (in Section 4).

2 Algorithm ME and Its Analysis

In this section, we introduce a deterministic memoryless online algorithm **ME** to schedule packets with deadlines in a single finite capacity queue. **ME** stands for “Modified EDF”. We first discuss the intuitions behind the algorithm **ME**. Then, we present **ME** and its analysis.

2.1 Intuitions of designing ME. We commence our study at a well-known real-time scheduling algorithm called **EDF** (“Earliest-Deadline-First”). **EDF** is one of the most important (and ever analyzed) dynamic priority algorithm, and the priority of a job (or a packet) is inversely proportional to its absolute deadline. In each time step, **EDF** schedules the packet with the earliest deadline [10]. The following example shows that even **EDF** calculates the best schedule sequence among all pending packets, it does not have a constant competitive ratio for the finite queue model ⁴. In this example, the queue capacity is $b \in \mathbb{Z}^+$ and we use (w, d) to represent a packet with value w and deadline d . We use $d = \infty$ to denote a packet with a very large deadline.

EXAMPLE 1. Initially, the algorithm’s queue is empty. In the first time step, $b - 1$ packets of (ϵ, i) , $i = 1, 2, \dots, b - 1$ and one packet $(1, \infty)$ arrive. In each of the following $b - 2$ time steps $2, \dots, b - 1$, only one packet $(1, \infty)$ is released to the queue.

When a new packet arrives, **EDF** tries to accept it and drops the minimum-value packet only if the queue is overflow. **EDF** sends the earliest-deadline packet in each time step. In our instance, **EDF** sends the packet (ϵ, i) in each time step $i = 1, 2, \dots, b - 1$ and all released packets $(1, \infty)$ are stored in its queue till the end of step $b - 1$. On the contrary, the optimal offline algorithm sends one packet $(1, \infty)$ in each of the first $b - 1$ time steps, and only one packet $(1, \infty)$ remains in its queue at the end of step $b - 1$.

⁴Such an algorithm is called *non-competitive*.

At the beginning of step b , EDF's queue is full and has b packets $(1, \infty)$. Now b packets of $(1 - \epsilon, 2 \cdot b)$ arrive but since they have weights smaller than the packets already in the buffer, they are dropped. Notice that EDF has lost $b - 1$ packets of $(1 - \epsilon, 2 \cdot b)$ due to the overflow happening in step b . At the beginning of each step $b + 1, b + 2, \dots, 2 \cdot b$, one packet $(\epsilon, b + i)$ ($i = 1, 2, \dots, b$) arrives. Given that all packets already in the queue keep their deadlines ∞ , EDF sends packet $(\epsilon, b + i)$ in each time step. On the contrary, the offline optimal algorithm sends $(1 - \epsilon, 2 \cdot b)$ in each step.

At the beginning of step $2 \cdot b + 1$, b packets of $(1 - \epsilon, 3 \cdot b)$ arrive and EDF's queue has a packet $(1 - \epsilon, 3 \cdot b)$ and $b - 1$ packets $(1, \infty)$. EDF sends a packet $(1 - \epsilon, 3 \cdot b)$ in step b . Notice that EDF has lost $b - 1$ packets of $(1 - \epsilon, 3 \cdot b)$ due to the overflow happening in step $2 \cdot b + 1$. At the beginning of each time step $2 \cdot b + 2, 2 \cdot b + 3, \dots, 3 \cdot b$, one packet $(\epsilon, 2 \cdot b + i)$ ($i = 2, 3, \dots, b$) arrives. Given that all packets already in the queue have deadline ∞ , EDF sends the packet $(\epsilon, 2 \cdot b + i)$ and keeps those in the queue without worrying about them being expired, hoping to send them in the future. On the contrary, the offline optimal algorithm sends $(1, \infty)$ in each time step.

We repeat this pattern. In the interval between 2 overflows (a period of b time steps), the optimal offline algorithm sends b large-value packets with value $1 - \epsilon$ and EDF sends only one large-value packet and $b - 1$ small value packets with value ϵ . We find that EDF cannot achieve a total value more than $1/b$ of what an offline optimal algorithm does. Assume we run n rounds of the same pattern of released packets. EDF sends all packets $(1, \infty)$ after n rounds. The competitive ratio c of EDF is (assume $n \cdot \epsilon = 1$)

$$c = \frac{1 \cdot b + (1 - \epsilon) \cdot b \cdot n + \epsilon \cdot b}{[\epsilon \cdot (b - 1) + 1] \cdot n + 1 \cdot b} \geq \frac{b}{1 + (2 \cdot b - 1)/n} \geq b - \frac{1}{1 + n/(2 \cdot b - 1)}.$$

Given b is large and if we repeat above pattern for at least $1/\epsilon$ times, EDF is not competitive in scheduling packets with deadlines in the finite capacity queue because the competitive c is not bounded by a constant. \square

Example 1 reveals that *even EDF keeps the set of pending packets with the maximum total value in each time step, it is not competitive*. The underlying idea of using EDF is that we do not drop any packet p unless p is going to expire at time d_p or in the queue, there are more packets with no less value than w_p having to be sent before d_p . The non-competitiveness of EDF over the above instance implies that we need a better method to identify whether a more valuable packet should be sent even well before its “real deadline”. For example, packet $(1 - \epsilon, \infty)$ released in step $b + 1$ in Example 1 should be sent early instead of being overflowed by later packets. Thus, it is critical for us to define and associate a “virtual deadline” with each packet, instead of the real deadline assigned, to denote the “best latest time” by which a packet should be sent. Inspired by the EDF instance in Example 1, we propose an algorithm ME, which keeps track of a packet's importance with respect to the others by using its “virtual deadline”.

2.2 Algorithm ME. At first, we introduce some notation. A packet p arrives at an integer time $r_p \in \mathbb{Z}^+$. p has a non-negative value $w_p \in \mathbb{R}^+$ and an integer deadline $d_p \in \mathbb{Z}^+$. Given a time t , we denote the buffer of an algorithm A to be Q_t^A . All buffer slots in Q_t^A are indexed as $0, 1, \dots, b - 1$. We use $Q_t^A(i)$ to denote the packet in the buffer slot indexed as i . If there is no packet in a buffer slot i , $Q_t^A(i)$ is a *null packet*. We associate each packet p a *virtual deadline* t_p . At p 's arrival, t_p is initialized as its real deadline specified by the adversary in the input sequence.

Given a set of pending packets, a *provisional schedule* specifies which packet should be sent in which time step no later than its *virtual deadline*, assuming no future arrivals. Given a set of pending packets

with virtual deadlines, an *optimal provisional schedule* is the one that achieves the maximum total value of packets among all provisional schedules on pending packets. The optimal provisional schedule gives a greedily optimal schedule of all the pending packets at time t : If there is no future arrivals, sending the packets each in one time step following the optimal provisional schedule is optimal for maximizing the total gain.

We develop our algorithm **ME** from our considerations on the **EDF** instance in Example 1. **ME** consists of 3 parts:

1. Based on the virtual deadlines of packets, calculate the optimal provisional schedule of sending the pending packets in the queue (including the new arrival in this time step), assuming there is no future arrivals (see Algorithm 1).
2. Update the virtual deadlines of packets, if needed (see Algorithm 2).
3. Send the packet with the earliest virtual deadline *or* the maximum-value packet, based on the ratio of these two packets (see Algorithm 2).

Assume for each packet p , we have known its virtual deadline t_p . The following procedure **OPS** (**OPS** stands for “Optimal Provisional Schedule”) greedily calculates the optimal provisional schedule from the set of pending packets \mathbf{S} at time t . In **OPS**, we first sort packets in non-increasing weight order. Then we pick up a packet p and put it into an empty queue as later as we could. If we cannot find such an empty buffer slot for p , this packet is discarded. All packets selected to be put into the queue are claimed to be in the optimal provisional schedule. **OPS** is described in Algorithm 1.

Algorithm 1 **OPS**(\mathbf{S}, t)

- 1: Sort all packets in \mathbf{S} in non-increasing weight order, with ties broken in favor of the larger virtual deadlines.
 - 2: **while** $\mathbf{S} \neq \emptyset$ **do**
 - 3: Pick up a packet p from \mathbf{S} .
 - 4: **for** each buffer slot i indexed from $\min\{t_p - t, b - 1\}$ down to 0 **do**
 - 5: **if** there is no packet in the buffer slot indexed as i **then**
 - 6: Put p into the i -th buffer slot.
 - 7: Remove p from \mathbf{S} .
 - 8: **Break.**
 - 9: **end if**
 - 10: **end for**
 - 11: **if** p is not added into the queue **then**
 - 12: Discard p .
 - 13: **end if**
 - 14: **end while**
 - 15: Sort all packets in the queue in non-decreasing virtual deadline order, with ties broken in favor of the larger value packets.
-

Using an interchange argument, we prove the optimality of **OPS**.

LEMMA 2.1. ***OPS**(\mathbf{S}) calculates the optimal provisional schedule for a set of pending packets \mathbf{S} .*

Proof. In the algorithm OPS, for each packet p , we either move p into the queue or we permanently discard it. We finalize the provisional schedule \tilde{S} in a greedy manner. To prove Lemma 2.1, it is sufficient to prove that for *each* packet p in the optimal provisional schedule S^* , \tilde{S} and S^* choose the same set of packets to put into buffer slots $[\tilde{S}(p) - t, b - 1]$, where $\tilde{S}(p)$ denotes the time step in which p is put into $\tilde{S}(p)$, given the set of pending packets and the assumption of no future arrivals. Without loss of generality, we assume all packets in S^* are sorted in non-decreasing deadline order, with ties broken in favor of the larger value packets.

We assume there exists an optimal provisional schedule S^* . If $\tilde{S} = S^*$, Lemma 2.1 holds immediately. Let us assume $\tilde{S} \neq S^*$. We then compare the packets scheduled in \tilde{S} and S^* from the buffer slot indexed as $b - 1$ in reverse order. q is the first packet appearing in the schedule S^* that is different from its counterpart in \tilde{S} (in the backward manner) and the corresponding time slot in \tilde{S} contains p . A packet $p \neq q$ must be found.

We apply the interchange argument to prove $\tilde{S} = S^*$. From our procedure of selecting packets in OPS, we know that any packet in a queue from $\tilde{S}(p)$ to its virtual deadline t_p has a value larger than or equal to w_p .

1. If $w_q > w_p$, then, q should be chosen before p when we create \tilde{S} and q should be put in the position $\tilde{S}(p)$ instead of the position of p .
2. If $w_q \leq w_p$, the optimal provisional schedule S^* should contain p since it includes q (t_p and t_q are not before the time slot S^* schedules q). Without losing any value, S^* can swap p and q since p is not in any buffer slot from $S^*(p)$ to t_p .

Thus, in this step, both S^* (after swapping p and q) and \tilde{S} schedule the same packet p . Lemma 2.1 is proved. ■

Now, we present the algorithm ME. ME consists of maintaining packets in the queue (including selecting packets and updating their virtual deadlines) and delivering a packet at the end of each time step. For each new arrival p , its virtual deadline t_p is initialized as its real deadline d_p . If there are more than one packet arriving, we consider them one by one. The deadline d_p is specified by the adversary at its arrival. Then we calculate all existing packets in the queue and p to find the optimal provisional schedule from time t . After we obtain the optimal provisional schedule, we update some packets' virtual deadlines, if necessary. Each packet updates its virtual deadline to the tentative time step specified in the optimal provisional schedule. At last, we send either the packet with the earliest virtual deadline (if it has a sufficient large value), or the maximum-value packet (otherwise). ME is described in Algorithm 2.

Directly from the algorithm ME, for each $p \in Q_t^{\text{ME}}$, we conclude the following properties of its virtual deadline t_p :

REMARK 1. All packets $p \in Q_t^{\text{ME}}$ have their t_p sorted in strictly increasing order as $t, t + 1, \dots, t + |Q_t^{\text{ME}}| - 1$, where $|Q_t^{\text{ME}}|$ is the number of packets in the queue. So, unless a new arrival p comes with its virtual deadline $t_p = d_p > t + |Q_t^{\text{ME}}| - 1$, accepting p will lead to dropping exactly one packet in Q_t^{ME} .

REMARK 2. Every time when t_p is updated (if any), t_p is decreased strictly. For any packet $p \in Q_t^{\text{ME}}$, $r_p \leq t_p \leq d_p$.

REMARK 3. All packets in the buffer have distinct virtual deadlines, which may not be the same as their deadlines specified in the input sequence.

Algorithm 2 ME(S, t)

- 1: For each new arrival p , set $t_p = d_p$.
 - 2: Calculate the optimal provisional schedule S^* by running $\text{OPS}(Q_t^{\text{ME}} \cup p, t)$.
 - 3: Drop all packets not in S^* .
 - 4: Update the virtual deadline t_j of a packet $j \in S^*$ as $t + i$ ($\leq d_j$), where i is the index of the buffer slot that j is residing in the optimal provisional schedule queue.
 {Notice that $Q_t^{\text{ME}}(i) = j$.}
 {Let the packet with the earliest virtual deadline be e , let the maximum-value packet be h , with ties broken in favor of the earliest virtual deadline.}
 - 5: **if** $w_e \geq w_h/\alpha$ **then**
 - 6: Send e .
 - 7: **else**
 - 8: Send h .
 - 9: **end if**
 {Lines 5 to 9 are as in EDF_α [12].}
-

2.3 Analysis of ME.

THEOREM 2.1. *ME is a deterministic 3-competitive algorithm for scheduling packets with deadlines in the finite queue model, where the parameter α in the algorithm is set 2.*

Fix an input sequence of arriving packets. The actions of the algorithm can be regarded as a sequence of *packet arrival events* and *packet delivery events* $\tau := \tau_1 \tau_2 \dots$. Then, in our algorithm **ME** and its analysis, if not mentioning, we use the subscript t to denote the event τ_t , instead of the time step t . A single time step may involve more than one arrival events and only one delivery event.

In analyzing online algorithms, potential function approach and charging scheme are two commonly used methods [9]. The potential function method assigns some values as the potentials to the online algorithm and the adversary's configurations respectively, and then compares the change of the potentials in each time step to bound the competitive ratio. In our analysis, we use a modified potential function to prove Theorem 2.1. We let **ADV** denote the adversary of **ME** and \mathbb{O} denote the set of packets sent by **ADV**, i.e., the packets in the optimal solution. Without loss of generality, we assume **ADV** sends the earliest deadline packet in each time step. Let the packets sent by **ADV** be $p_1, p_2, \dots, p_i, \dots$ in order. A packet $p_i \in \mathbb{O}$ is delivered in step i . If there is no packet to send in step t , p_t is a *null* packet. Our analysis (especially for packet arrivals) depends on a critical observation on the adversary and a property of **ME**:

REMARK 4. *Assume in steps 1, 2, ..., i, n, ADV sends packets $p_1, p_2, \dots, p_i, \dots, p_n$ in order. Clearly, $r_{p_i} \leq i \leq d_{p_i}$. Furthermore, in ADV's queue, we are free to modify the packets' deadlines d_{p_i} to t'_{p_i} as long as $r_{p_i} \leq i \leq t'_{p_i}$.*

REMARK 5. *For any packet j in ME's queue, the minimum value of a packet i with $t_i \leq t_j$ does not decrease over time.*

We use Φ_t^{ME} (respectively, Φ_t^{ADV}) to denote the potential of the queue of **ME** (respectively, **ADV**) at time t . Φ_t^{ME} (respectively, Φ_t^{ADV}) is the sum of the (mapped) \mathbb{O} -packets (respectively, \mathbb{O} -packets) in the queue.

Define S_t^A as the packet sent by an algorithm A. Our goal is to prove that at the end of each event, the following inequality

$$3 \cdot \sum_{j \in S_t^{\text{ME}}} w_j + \Phi_t^{\text{ME}} \geq \sum_{k \in S_t^{\text{ADV}}} w_k + \Phi_t^{\text{ADV}}, \quad (2.1)$$

holds. As a consequence, this yields Theorem 2.1.

Let Δ_t^{ME} (respectively, Δ_t^{ADV}) denote the difference of the left (respectively, right) side of Inequality 2.1 from time $t - 1$ to time t , i.e.,

$$\Delta_t^{\text{ME}} := 3 \cdot \sum_{i \in (S_t^{\text{ME}} \setminus S_{t-1}^{\text{ME}})} w_i + \Phi_t^{\text{ME}} - \Phi_{t-1}^{\text{ME}}, \quad (2.2)$$

$$\Delta_t^{\text{ADV}} := \sum_{k \in (S_t^{\text{ADV}} \setminus S_{t-1}^{\text{ADV}})} w_k + \Phi_t^{\text{ADV}} - \Phi_{t-1}^{\text{ADV}}. \quad (2.3)$$

Obviously, Inequality 2.1 holds before the first event since packets have not been sent so far. In order to prove Theorem 1, it is sufficient to prove that for each event, the following inequality holds since it leads to Inequality 2.1.

$$\Delta_t^{\text{ME}} \geq \Delta_t^{\text{ADV}}. \quad (2.4)$$

In order to prove Inequality 2.4, we present a set of invariants which hold at the end of each event.

I_1 . $\Delta_t^{\text{ME}} \geq \Delta_t^{\text{ADV}}$.

I_2 . ADV's queue contains only the set of packets it will send. For each packet $j \in (Q_t^{\text{ME}} \cap Q_t^{\text{ADV}})$, ADV has the virtual deadline t_j as its real deadline.

For each packet $j \in Q_t^{\text{ME}}$, j maps to at most one packet $j' \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{ME}})$. For each packet $p' \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{ME}})$, p' must be mapped uniquely by a packet $p \in Q_t^{\text{ME}}$.

I_3 . If $j \in Q_t^{\text{ME}}$ maps to $j' \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{ME}})$, for any packet $i \in Q_t^{\text{ME}}$ with $t_i \leq t_j$, the following inequalities are true: $t_i \leq d_{j'}$ and $w_i \geq w_{j'}$.

We prove that the set of invariants hold separately for both the events of packet arrivals and packet deliveries. Summing these inequalities over the arrivals and deliveries happened in one single time step yields the claim for a single time step; summing over all time steps proves Theorem 2.1. To prove the existence of the above set of invariants, we apply case study in the following.

Proof. We show the set of invariants hold at the end of each time step. For each time step, we consider packet arrivals and packet delivery separately. In the following case analysis, we update the packets in ADV's queue as well as their mappings to the packets in ME's buffer.

If not mentioned otherwise, everything else remains unchanged at the end of this event. For ease of presentation, we assume buffer slots are indexed as 1, 2, ..., b . We use \mathbb{O} to denote the set of packets sent by the adversary. Let e and h denote the packets with the earliest virtual deadline and the packet with the maximum value, with ties broken in favor of the earliest virtual deadline one. Remember all packets in the queue have distinct virtual deadlines (see Algorithm 1 and Remark 3).

We are going to show that in each time step, the ratio of ADV's gain over ME's gain is bounded by $1 + \alpha$, or $1 + 3/\alpha$, or $2 + 2/\alpha$. The competitive ratio 3 is optimized at $\alpha = 2$ for

$$\min \max\{1 + \alpha, 1 + 3/\alpha, 2 + 2/\alpha\}. \quad (2.5)$$

2.3.1 Packet delivery In each time step, ME either sends e or h . If ME sends e , $w_e \geq w_h/\alpha$; otherwise, ME sends h . We assume ADV sends j . At the end of this delivery event, e is out of ME's queue because of its virtual deadline. We summarize all the possible consequences into the following 5 cases, based on the packet ME sends and the packet ADV sends in each step.

1. Assume ME and ADV send the same packet j .

We charge ME w_j . We charge ADV w_j initially. If $j \neq e$, i.e., $j = h$ and we know $w_h > \alpha \cdot w_e$. We charge ADV more $w_e + w_{e'} + w_{j'}$, where e' is the packet mapped by e and j' is the packet mapped by j , if any. From the invariant I_3 , $\max\{w_{e'}, w_{j'}\} \leq w_e$. Thus, the ratio of the modified gain for ADV and ME is bounded by $\max\{(w_e + w_{e'})/w_e, (w_e + w_{e'} + w_j + w_{j'})/w_j\} = \max\{2, 1 + 3/\alpha\} = 1 + 3/\alpha$, where $\alpha = 2$.

2. Assume ME sends e and ADV sends $j \notin Q_t^{\text{ME}}$.

From the invariant I_2 , we assume $j = p'$, and j is mapped by a packet $p \in Q_t^{\text{ME}}$. Since ME sends e , $w_e \geq w_h/\alpha$.

- (a) Assume $e = p$.

We charge ME w_e and charge ADV $w_j \leq w_p = w_e$. Then the ratio of the modified gains is $w_j/w_e \leq 1$.

- (b) Assume $e \neq p$ and e is not in any mapping.

We charge ME w_e and we charge ADV $w_e + w_j$ (given e possibly being an \mathbb{O} -packet). Then the ratio of the modified gains is $(w_e + w_j)/w_e \leq (w_e + w_h)/w_e \leq 1 + \alpha$.

- (c) Assume $e \neq p$ and e maps $e' \in Q_t^{\text{ADV}}$.

We have $d_{e'} > d_j$, otherwise, ADV will select e' to send (because we assume ADV selects packets to send in the earliest deadline order). Thus, $d_{e'} \geq d_j = d_{p'} \geq t_p$ (the third inequality holds because of the invariant I_2). Also, $p \notin Q_t^{\text{ADV}}$, otherwise, since $t_p = d_p \leq d_j$, ADV will send p instead of j .

We charge ME w_e and we charge ADV $w_{e'} + w_j$. Then the ratio of the modified gains is $(w_{e'} + w_j)/w_e \leq (w_e + w_h)/w_e = 1 + \alpha$.

3. Assume ME sends e and ADV sends $j \in Q_t^{\text{ME}}$, $e \neq j$.

$w_e \geq w_h/\alpha \geq w_j/\alpha$. $e \notin Q_t^{\text{ADV}}$, otherwise, ADV will send e instead of j in this step.

We charge ME w_e . We charge ADV $w_{e'} + w_j$, assuming e maps e' . Then the ratio of the modified gains is $(w_{e'} + w_j)/w_e \leq 1 + \alpha$. If j maps a packet in ADV's queue only, this mapping still holds at the end of this delivery event.

4. Assume ME sends $h \neq e$ and ADV sends $j \notin Q_t^{\text{ME}}$.

Note $w_e < w_h/\alpha$. From the invariant I_2 , we can assume $p \in Q_t^{\text{ME}}$ maps $p' = j \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{ME}})$. Since ADV sends p' and $d_{p'} \geq t_p$ (from the invariant I_2), we know $p \notin Q_t^{\text{ADV}}$. Thus, $w_j = w_{p'} \leq w_e$. h should be in Q_t^{ADV} , otherwise, ADV can send h instead of j to gain more value in this time step. Also, $w_{h'} \leq w_e$.

(a) Assume $e = p$.

e is out of ME's queue at the end of this event because of its virtual deadline expires. Then we charge ME w_h . We charge ADV $w_e + w_{e'} + w_h + w_{h'}$, assuming e maps $e' = j$ and h maps h' . Note $\max\{w_{e'}, w_{h'}\} \leq w_e < w_h/\alpha$. Then the ratio of the modified gains is $(w_e + w_{e'} + w_h + w_{h'})/w_h \leq (3 + \alpha)/\alpha = 1 + 3/\alpha$.

(b) Assume $e \neq p$.

Assume $e \neq p$ and e maps $e' \in Q_t^{\text{ADV}}$. We have $d_{e'} > d_j$, otherwise, ADV will select e' to send (because we assume ADV selects packets to send in earliest deadline order). Thus, $d_{e'} \geq d_j = d_{p'} \geq t_p$.

We map p to e' . We charge ME w_h and we charge ADV $w_h + w_{h'} + w_e + w_{e'}$. Then the ratio of the modified gains is $(w_e + w_{e'} + w_h + w_{h'})/w_h \leq 1 + 3/\alpha$.

5. Assume ME sends $h \neq e$ and ADV sends $j \in Q_t^{\text{ME}}$, $j \neq h$.

Clearly, $t_j = d_j < t_h = d_h$, otherwise, ADV can always swap the sending sequences of j and h . h should be in Q_t^{ADV} , otherwise, ADV can send h instead of j to gain more value in this time step. Also, $w_{h'} \leq w_e$.

(a) Assume $j = e$.

We charge ME w_h and charge ADV $w_h + w_{h'} + w_e + w_{e'}$, assuming e maps e' and h maps h' . $\max\{w_{e'}, w_{h'}\} \leq w_e \leq w_h/\alpha$. Then the ratio of the modified gains is $(w_h + w_{h'} + w_e + w_{e'})/w_h \leq 1 + 3/\alpha$.

(b) Assume $j \neq e$.

e is not an \mathbb{O} -packet. Assume e maps e' in ADV's queue; if e' does not exist, we let e' be a null packet. $w_{e'} \leq w_e$. Note $w_{j'} \leq w_e$. At the end of this delivery, j is still in ME's queue but j is not in ADV's queue.

We charge ME w_h . Then, we charge ADV $w_h + w_{h'} + w_j + w_{e'}$. Then the ratio of the modified gains is $(w_h + w_{h'} + w_j + w_{e'})/w_h \leq (w_h + w_e + w_h + w_e)/w_h = 2 + 2/\alpha$.

2.3.2 Packet arrivals Remember that from the properties of algorithm ME (see Remark 1 and Remark 2), for each new arrival p , if admitting p results in a packet i leaving Q_t^{ME} , the total value of the queue is not decreasing. p can always be a candidate packet to map the packet which was mapped by the packet evicted due to accepting p (since $w_p \geq w_i$). Also, the slack time of p , defined as $d_p - t$, is no larger than the total number of packets in the queue $|Q_t^{\text{ME}}|$, otherwise, p will be accepted by ME without evicting a packet (see Algorithm 1).

For each arriving event at time t , $S_{t+1}^{\text{ME}} \setminus S_t^{\text{ME}} = Q_{t+1}^{\text{ADV}} \setminus Q_t^{\text{ADV}} = \emptyset$. Thus, for each new arrival event, we only need to consider the change of mappings, if any.

From the property of the adversary, we know that all packets in ADV's queue (supposed to be sent by ADV) will not be evicted when we put new \mathbb{O} -packets in ADV's queue. Let us consider the case when introducing an \mathbb{O} -packet p results in a packet i leaving ME's queue. If i is not in ADV's queue, we are fine with all mappings and potentials because there is no loss to Φ_t^{ME} . We only consider the case when i is in ADV's queue.

Assuming i is an \mathbb{O} -packet in ADV's queue, we first claim that we can always find a packet q , which is not in ADV's queue with $w_q \geq w_i$. Otherwise, ADV does not accept p as well. Then we collect all

\mathbb{O} -packets in ME's queue but with deadlines $\leq |Q_t^{\text{ME}}|$, as they are need to be delivered by ADV by time $t + |Q_t^{\text{ME}}|$, it does not hurt to assign them deadlines in strictly decreasing order from $t + |Q_t^{\text{ME}}|$. Therefore, the evicted \mathbb{O} -packet i can be assigned a deadline as the virtual deadline of the latest non- \mathbb{O} -packet in a buffer slot no later than $d_i - t$ in ME's queue. Let this packet be j . We can remove i from ADV's queue and put j with t_j as its deadline and w_j as its value in ADV's queue. These operations do not hurt ADV because of Remark 5. Above reasoning can also be applied to the case when a new \mathbb{O} -packet p is rejected by ME.

Based on our case study at packet arrival events and packet delivery events discussed above, Theorem 2.1 is proved. ■

Some side research results on the finite queue model are shown as follows; they give the upper and lower bounds that (some) online algorithms can achieve.

We use \mathbf{S}_t to denote both the provisional schedule for time steps $[t, +\infty)$ and the set of packets specified by the schedule. All known online algorithms for the bounded-delay model [17], [15], [11], [20], [21], [13] calculate their optimal provisional schedules at the beginning of each time step. These algorithms only differ by the packets they select to send. The deterministic online algorithms in such a broad family are defined as the *best-effort admission algorithms*.

DEFINITION 1. Best-effort admission algorithm. *Consider an online algorithm ON and a set of pending packets \mathbf{P}_t at time t . If ON calculates the optimal provisional schedule \mathbf{S}_t on \mathbf{P}_t and selects one packet from \mathbf{S}_t to send in the step t , we call ON a best-effort admission algorithm.*

THEOREM 2.2. *The lower bound of competitive ratio for the best-effort admission algorithms is 2.*

Proof. In the following instance, we will show: *If the buffer size is bounded, the packets that the optimal offline algorithm chooses to send may not be from the optimal provisional schedule calculated by the online algorithm, even if both algorithms have the same set of pending packets.* This property does not hold in the bounded-delay model; and it leads that any deterministic best-effort admission algorithm cannot achieve a competitive ratio better than 2.

Assume the buffer size is b . Let a best-effort admission online algorithm be ON. We use (w_p, d_p) to represent a packet p with a value w_p and a deadline d_p . Initially, the buffer is empty. A set of packets, from which the optimal offline algorithm will accept $b - 1$ packets from them and eventually send, are released: $(1, b + 1), (1, b + 2), \dots, (1, b + b)$. Based on its definition, upon these packets' arrival, ON will select all of them to put into its buffer. Notice that all packets released have deadlines larger than the buffer size b . The optimal offline algorithm drops $(1, b + 1)$, and keeps $(1, b + 2), \dots, (1, b + b)$ in its buffer.

In the same time step, b packets $(1 + \epsilon, 1), (1 + \epsilon, 2), \dots, (1 + \epsilon, b)$ are released afterwards. There are no more new packets arriving in this step. The optimal offline algorithm only accepts $(1 + \epsilon, 1)$. Thus, after processing arrivals in step 1, the optimal offline algorithm send the packet $(1 + \epsilon, 1)$. Instead, ON calculates the optimal provisional schedule in step 1 which includes all these newly arriving packets with value $1 + \epsilon$. All such packets will be accepted by ON, but the packets $(1, b + i), \forall i = 1, 2, \dots, b$, will be dropped due to the buffer size constraint. ON sends a packet with value $1 + \epsilon$ in the first step.

At the beginning of each step $i = 2, 3, \dots, b$, only one packet $(1 + \epsilon, i)$ is released. At the end of step b , no new packets will be released in the future. Since the time after the first step, all packets

available to ON have their deadlines $\leq b$. Thus, ON cannot schedule sending packets with a total value $\geq (1 + \epsilon) \cdot (b - 1)$ in the time steps 2, 3, ..., b .

Of course, since there is one empty buffer slot at the beginning of each time step $i = 2, 3, \dots, b$, the optimal offline algorithm can accept and send all newly released packets $(1 + \epsilon, i)$ in steps $i = 2, 3, \dots, b$. At the end of step b , the packets $(1, b + 2), (1, b + 3), \dots, (1, b + b)$ are still remained in the optimal offline algorithm's buffer (they are not in ON's buffer though). Since there is no future arrivals, these $b - 1$ packets will be transmitted eventually by the optimal algorithm in the following $b - 1$ steps. The total value of ON achieves is $(1 + \epsilon) \cdot b$ while the optimal offline algorithm gets a total value $(1 + \epsilon) \cdot b + 1 \cdot (b - 1)$. The competitive ratio for this instance is

$$c = \frac{(1 + \epsilon) \cdot b + 1 \cdot (b - 1)}{(1 + \epsilon) \cdot b} = 2 - \frac{1 + b \cdot \epsilon}{b + b \cdot \epsilon} \geq 2 - \frac{2}{b}, \quad \text{if } \epsilon \cdot b = 1 \text{ and } b \geq 2.$$

If b is large, ON cannot perform asymptotically better than 2-competitive. This loss is due to ON calculating optimal provisional schedule to find out the packet to send in each time step. Theorem 2.2 is proved. ■

LEMMA 2.2. *The simple greedy algorithm, which selects packets in the optimal provisional schedule and sends the maximum-value packet in each time step, is no better than 4-competitive.*

Proof. In the following instance, we will show that the greedy algorithm **Greedy**, which calculates the optimal provisional schedule and schedules the maximum-value packet, cannot be better than 4-competitive. Let the buffer size be b . Without loss of generality, we assume b is even. We use (w_p, d_p) to represent a packet p with a value w_p and a deadline d_p .

Suppose at the end of step 0, the buffer is empty. A set of packets, which the optimal offline algorithm will eventually send, are released: $(1, b + 1), (1, b + 2), \dots, (1, b + b)$. Notice that all packets in the buffer have deadlines larger than the buffer size b .

At the beginning of step 1, b packets $(1 + 1 \cdot \epsilon, 1), (1 + 2 \cdot \epsilon, 2), \dots, (1 + b \cdot \epsilon, b)$ are released. **Greedy** accepts all these newly arriving packets. The optimal offline algorithm only accepts $(1 + \epsilon, 1)$, drops $(1, b + 1)$, and keeps $(1, b + 2), \dots, (1, b + b)$ in its buffer. In step 1, the optimal offline algorithm send $(1 + \epsilon, 1)$. Instead, **Greedy** will accept all newly released packets in step 1, thus, all packets $(1, b + i)$ for any $i = 1, 2, \dots, b$ are dropped. **Greedy** sends the packet $(1 + b \cdot \epsilon, b)$. At the end of this step, the packet $(1 + 1 \cdot \epsilon, 1)$ in **Greedy**'s buffer expires.

At the beginning of each step $i = 2, 3, \dots, b$, only one packet $(1 + \epsilon, i)$ is released. At the end of step b , no future packets will be released. **Greedy** rejects all these newly released packets. **Greedy** will send the packets $(1 + (b - 1) \cdot \epsilon, b - 1), (1 + (b - 2) \cdot \epsilon, b - 2), \dots, (1 + (b/2 + 1) \cdot \epsilon, b/2 + 1)$ in the following $b/2 - 1$ time steps. All the packets $(1 + 2 \cdot \epsilon, 2), (1 + 3 \cdot \epsilon, 3), \dots, (1 + (b/2) \cdot \epsilon, b/2)$ will be dropped due to their deadlines.

Of course, the optimal offline algorithm can send all newly released packets in steps 2, 3, ..., b . At the end of step b , the packets $(1, b + 2), (1, b + 3), \dots, (1, b + b)$ are still remained in the optimal offline algorithm's buffer, but not in **Greedy**'s buffer. Since there is no future arrivals, these $b - 1$ packets will be transmitted eventually by the optimal algorithm in the following $b - 1$ steps. If b is large, **Greedy** cannot perform better than 4-competitive. The loss is due to **Greedy**'s first step in which optimal provisional schedule is used in selecting packets in the buffer and its greedy manner in sending packets in the first $b/2$ time steps. Lemma 2.2 is proved. ■

3 Algorithm RME and Its Analysis

In this section, we present a randomized algorithm for the finite capacity queue model. Our algorithm is named RME, which stands for “Randomized ME”. Similar to ME, RME consists of two parts in handling packet arrivals and packet deliveries respectively in each time step. The difference is that RME employs a random variable to decide whether to send the earliest packet or the most valuable packet.

We would like to point out that even RME makes a random choice during its execution, this randomization is executed by the algorithm internally and has nothing to do with the characteristics of the input sequence. The adversary is allowed to generate the input sequence to maximize the competitive ratio. No stochastic assumption is made on the input sequence.

3.1 Algorithm RME. For each packet arrival event, RME works the same as what ME does. That is, RME calls $\text{OPS}(\mathbf{S}, t)$ to identify the packets in its buffer deterministically, where \mathbf{S} is the set of pending packets and t is the current time. In packet delivery, a random variable β is used to facilitate scheduling.

We use e to denote the packet with the earliest-virtual deadline packet and h to denote the earliest maximum value packet in the buffer. The algorithm works as follows. If e has a sufficiently large value with $w_e \geq w_h/\alpha$, we send e deterministically. Otherwise (i.e., $w_e < w_h/\alpha$), we choose β uniformly on $[0, 1]$. If $\beta \in [0, \gamma]$ (we will decide γ later. The parameter γ influences the competitive ratio of the algorithm), we deliver e , otherwise (i.e., if $\beta \in (\gamma, 1]$ and $w_e < w_h/\alpha$), we deliver h . The pseudo code of RME is described in Algorithm 3, where the set of pending packets at time t is \mathbf{S} .

Algorithm 3 RME(\mathbf{S}, t)

- 1: For each new arrival p , set $t_p = d_p$.
 - 2: Calculate the optimal provisional schedule S^* by running $\text{OPS}(Q_t^{\text{ME}} \cup p, t)$.
 - 3: Drop all packets not in S^* .
 - 4: Update the virtual deadline t_j of a packet $j \in S^*$ as $t + i$, where i is the index of the buffer slot that j is in.
 - 5: **if** $w_e \geq w_h/\alpha$ **then**
 - 6: Send e .
 - 7: **else**
 - 8: Choose β uniformly on $[0, 1]$.
 - 9: **if** $\beta \in [0, \gamma]$ **then**
 - 10: Send e .
 - 11: **else**
 - 12: Send h .
 - 13: **end if**
 - 14: **end if**
-

3.2 Analysis of RME.

THEOREM 3.1. *RME is a randomized ($\phi^2 \approx 2.618$)-competitive algorithm for scheduling packets with deadlines in the finite queue model, where $\alpha = \phi \approx 1.618$ and $\gamma = 1/\phi^2 \approx 0.382$.*

Some formulas are used in our analysis:

$$1/\phi + 1 = \phi, \quad \phi + 1 = \phi^2, \quad 2 + 1/\phi = \phi^2, \quad \phi + 1/\phi^2 = 2.$$

Proof. At first, we examine the expected gain that the algorithm RME can gain in a time step. From the algorithm itself, we know that RME gains,

$$\begin{aligned} w_e, & \quad \text{if } w_e \geq w_h/\alpha, \\ w_e, & \quad \text{if } w_e < w_h/\alpha \text{ and } \beta \in [0, 1/\phi^2], \\ w_h, & \quad \text{if } w_e < w_h/\alpha \text{ and } \beta \in (1/\phi^2, 1]. \end{aligned}$$

In either way ($w_e \geq w_h/\phi$ or $w_e < w_h/\phi$), the algorithm gains an expected value of

$$\mathbf{E}(W_t) \geq \min\{w_h/\phi, w_e \cdot (1/\phi^2) + w_h \cdot (1 - 1/\phi^2)\} = w_h/\phi.$$

Remember S_t^A is the packet sent by an algorithm A. We observe the following set of invariants about ADV and RME's buffers:

$V_1.$

$$\phi^2 \cdot \mathbf{E}\left(\sum_{j \in S_t^{\text{RME}}} w_j\right) + \Phi_t^{\text{RME}} = \phi^2 \cdot \sum_{j \in S_t^{\text{RME}}} \mathbf{E}(w_j) \geq \sum_{k \in S_t^{\text{ADV}}} w_k + \Phi_t^{\text{ADV}},$$

where linearity of expectations is used in the first equality.

$V_2.$ ADV's queue contains only the set of packets it sends. For each packet $j \in (Q_t^{\text{RME}} \cap Q_t^{\text{ADV}})$, ADV has the virtual deadline t_j as this packet's modified real deadline d_j .

For each packet $j \in Q_t^{\text{RME}}$, j maps to at most one packet $j' \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{RME}})$. For each packet $p' \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{RME}})$, p' must be mapped uniquely by a packet $p \in Q_t^{\text{RME}}$.

$V_3.$ If $j \in Q_t^{\text{RME}}$ maps $j' \in (Q_t^{\text{ADV}} \setminus Q_t^{\text{RME}})$, for any packet $i \in Q_t^{\text{RME}}$ with $t_i \leq t_j$, $t_i \leq d_{j'}$ and $w_i \geq w_{j'}$.

Similar to the analysis of ME, packet arrival and delivery events are analyzed separated from case studies. We omit the analysis on packet arrival, which has been presented in the proof of Theorem 2.1. In the following, we discuss the invariants in the randomized packet delivery events.

In each time step, RME either sends e or h . If RME sends h , we must have $w_h \geq \alpha \cdot w_e = \phi \cdot w_e$ (from the algorithm). We assume ADV sends j . At the end of this delivery event, e is out of RME's queue because of its virtual deadline. We study the following cases which are categorized based on the packet RME sends and the packet ADV sends in each time step.

1. Assume RME sends e and $w_e \geq w_h/\phi$.

As what we have seen in the proof of Theorem 2.1, the ratio of the modified gain is bounded by $1 + \alpha = 1 + \phi = \phi^2$.

2. Assume RME sends e with $w_e < w_h/\phi$.

This case happens with a probability of $1/\phi^2$ when $w_e < w_h/\phi$. We combine this case with the next case to get the expected competitive ratio.

3. Assume RME sends $h \neq e$ and ADV sends $j \neq h$.

This case happens with a probability of $1/\phi$ when $w_e < w_h/\phi$. e leaves the buffer at the end of this delivery due to its virtual deadline. We assume $j \neq e$.

(If $j = e$, we just ignore all w_j and $w_{j'}$ in our following calculation and all inequalities still hold with simpler representations. If $j = h$, we simply use h to replace j in the following calculation and all inequalities hold also.)

We categorize the cases with e is in a mapping and e is not in a mapping.

If e is in a mapping, e is sent with a probability of $1/\phi^2$ and h is sent with a probability of $1/\phi$. In the first case, we charge $\text{ADV } w_e + w_{e'}$ where e maps e' . In the second case, we charge $\text{ADV } w_e + w_{e'} + w_j + w_{h'}$. Thus, the expected competitive ratio is

$$\begin{aligned}
c &= \frac{\text{ADV}_t}{\mathbf{E}(\text{RME}_t)} \\
&= \frac{(1/\phi^2) \cdot (2 \cdot w_e) + (1/\phi) \cdot (w_e + w_{e'} + w_j + w_{h'})}{(1/\phi^2) \cdot w_e + (1/\phi) \cdot w_h} \leq \frac{2w_e + \phi \cdot 2w_e + \phi \cdot 2w_h}{w_e + \phi \cdot w_h} \\
&\leq \frac{(2\phi + 2)(w_h/\phi) + 2\phi \cdot h}{w_h/\phi + \phi \cdot w_h} = \frac{2 + 2/\phi + 2\phi}{1/\phi + \phi} \leq \phi^2.
\end{aligned}$$

If e is not in a mapping, e is sent with a probability of $1/\phi^2$ and h is sent with a probability of $1/\phi$. In the first case, we charge $\text{ADV } w_j \leq w_h$. In the second case, we charge $\text{ADV } w_j + w_h + w_{h'}$. Remember $w_e \geq w_{h'}$ (see the invariant V_3). Thus, the expected competitive ratio is

$$\begin{aligned}
c &= \frac{\text{ADV}_t}{\mathbf{E}(\text{RME}_t)} \\
&= \frac{(1/\phi^2) \cdot w_j + (1/\phi) \cdot (w_j + w_h + w_{h'})}{(1/\phi^2) \cdot w_e + (1/\phi) \cdot w_h} \leq \frac{w_h/\phi^2 + (w_h + w_h + w_e)/\phi}{w_e/\phi^2 + w_h/\phi} \\
&\leq \frac{w_h + 2 \cdot \phi \cdot w_h + \phi \cdot w_e}{w_e + \phi \cdot w_h} \leq \frac{2 \cdot \phi + 1}{\phi} = \phi^2.
\end{aligned}$$

Based on the above analysis, Theorem 3.1 is proved. ■

4 The Optimal Offline Algorithm and Its Analysis

Let \mathcal{O} denote the set of packets sent by an optimal offline algorithm. Our algorithm is simple. Fix an input sequence \mathcal{I} . We start from a set of packets $\mathbf{S}_0 \subseteq \mathcal{I}$ such that all packets in $\mathbf{S}_0 = \mathbf{S}$ can be delivered successfully if we send them in increasing order of deadlines. Given a set of packets \mathbf{S} , we get the total gain of $W(\mathbf{S})$. If $\mathbf{S}_0 = \mathcal{I}$, the algorithm is optimal. If $\mathbf{S}_0 \neq \mathcal{I}$, then we study the set of packets $\mathcal{I} \setminus \mathbf{S}_0$. We sort all packets in $\mathcal{I} \setminus \mathbf{S}_0$ in increasing order of deadlines. For each packet j , adding j into \mathbf{S}_0 generates a new set \mathbf{S} ; this results at most one packet i not being sent successfully (still under the earliest deadline policy). Then we run into a loop to pick up a packet $i \in \mathbf{S}$ with $w_i < w_j$ and see whether $W(\mathbf{S} \cup \{j\} \setminus \{i\}) > W(\mathbf{S})$. (i can be a *null packet* such that $\mathbf{S} \cup \{j\} \setminus \{i\} = \mathbf{S} \cup \{j\}$). If $W(\mathbf{S} \cup \{j\}) > W(\mathbf{S} \cup \{j\} \setminus \{i\}) > W(\mathbf{S})$, we add j into \mathbf{S} . If $W(\mathbf{S} \cup \{j\}) < W(\mathbf{S} \cup \{j\} \setminus \{i\})$, we drop i . We iteratively move j out of $\mathcal{I} \setminus \mathbf{S}_0$ into \mathbf{S} until $\mathcal{I} \setminus \mathbf{S}_0$ is empty. We claim that the schedule we finally have is optimal. It depends on the following two theorems.

THEOREM 4.1. *Given a set of packets \mathbf{S} , if all packets can be sent by their deadlines with the buffer size constraint, we can always schedule them in increasing order of deadlines among all pending packets in the buffer.*

Proof. This is a standard result. ■

LEMMA 4.1. \mathbf{S}_0 is easy to be construct. We simply pick up the earliest deadline packet to send and in each time step, greedily accept packets.

THEOREM 4.2. If a set of packets \mathbf{S} can be delivered successfully, $\mathbf{S} \cup \{j\}$ results at most one packet unsuccessfully sent. We can pick up any packet i with $w_i < w_j$ as the candidate and schedule $\mathbf{S} \cup \{j\} \setminus \{i\}$. If $W(\mathbf{S} \cup \{j\} \setminus \{i\}) > W(\mathbf{S})$ but $W(\mathbf{S} \cup \{j\}) \leq W(\mathbf{S})$, $i \notin \mathbb{O}$.

Proof. This is due to the property of matriod. ■

THEOREM 4.3. The optimal offline algorithm runs in polynomial-time $O(n^3)$, where n is the size of the input sequence.

Proof. Given a set of m packets \mathbf{S} , sorting all packets in \mathbf{S} (in order of increasing deadlines) takes time $O(m \cdot \log m)$. For each packet not in \mathbf{S} but not discarded yet, we take time $O(m)$ to locate a packet i with value $w_i < w_j$ and then we schedule $\mathbf{S} \cup \{j\} \setminus \{i\}$ (note that i can be a null packets such that $\mathbf{S} \cup \{j\} \setminus \{i\} = \mathbf{S} \cup \{j\}$). It takes time $O(m)$ to get the total gain of a schedule. Thus, identifying whether to accept j or to discard j takes time $O(m^2)$. Let the set of packets in the input sequence be n with $n > m$. The total running time of our algorithm is bounded by $O(n \cdot n^2) = O(n^3)$. ■

5 Conclusions and Open Problems

In this paper, we present two online algorithms for scheduling weighted packets with hard deadlines in a finite capacity queue and we provide their theoretical competitive analysis. The model we study generalizes the extensively studied bounded-delay model for QoS buffer management. Our model has significant importance in real system design since it takes into account of the realistic bound on the size of the router queues. The deterministic memoryless algorithm we present is 3-competitive and the randomized memoryless algorithm we present is $(\phi^2 \approx 2.618)$ -competitive. Both algorithms provide the worst-case guarantees to robustly optimize our objective (maximizing the weighted throughput) without applying any stochastic assumptions over the packet traffic. We propose a novel analysis approach by updating packets' parameters in an online manner. Instead of using the real deadlines, we introduce virtual deadlines, which are updated over time to help us make the best decision on when to send the packets. The virtual deadlines are strictly decreased over time and they guarantee the hard deadlines are always satisfied. This idea can be applied in many other online and real-time problems.

Closing or shrinking the gap of [1.618, 2.618] between the lower bound and upper bound of competitive ratios for this model is an open problem. For a broad family of online algorithms, including all previously known research for the bounded-delay model, the lower bound is 2. Our algorithmic framework can be applied to the multi-buffer model [5]. Getting an algorithm better than 9.82-competitive for the the multi-buffer model is still an interesting open problem.

References

- [1] W. Aiello, Y. Mansour, S. Rajagopalan, and A. Rosen. Competitive queue policies for differentiated services. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 431–440, 2000.

- [2] W. Aiello, Y. Mansour, S. Rajagopalan, and A. Rosen. Competitive queue policies for differentiated services. *Journal of Algorithms*, 55(2):113–141, 2005.
- [3] S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. *SIAM Journal on Computing*, 35(2):278–304, 2005.
- [4] N. Andelman, Y. Mansour, and A. Zhu. Competitive queuing policies for QoS switches. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–770, 2003.
- [5] Y. Azar and N. Levy. Multiplexing packets with arbitrary deadlines in bounded buffers. *Lecture Notes in Computer Science (SWAT)*, pages 5–16, 2006.
- [6] Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 82–89, 2003.
- [7] Y. Azar and Y. Richter. The zero-one principle for switching networks. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 64–71, 2004.
- [8] N. Bansal, L. K. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 196–207, 2004.
- [9] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [10] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichy. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, 2006.
- [11] F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.
- [12] M. Chrobak, W. Jawor, J. Sgall, and T. Tichy. Improved online algorithms for buffer management in QoS switches. *ACM Transactions on Algorithms*, 3(4), Article number 50, 2007.
- [13] M. Englert and M. Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–218, 2007.
- [14] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, (4):392–403, 2001.
- [15] B. Hajek. On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In *Proceedings of 2001 Conference on Information Sciences and Systems (CISS)*, pages 434–438, 2001.
- [16] T. Itoh and N. Takahashi. Competitive analysis of multi-queue preemptive QoS algorithms for general priorities. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(5):1186–1197, 2006.
- [17] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal of Computing (SICOMP)*, 33(3):563–583, 2004.
- [18] A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proceedings of 13th Annual European Symposium on Algorithms (ESA)*, pages 361–372, 2003.
- [19] F. Li. Competitive scheduling of packets with hard deadlines in a finite capacity queue. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, 2009.
- [20] F. Li, J. Sethuraman, and C. Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802, 2005.
- [21] F. Li, J. Sethuraman, and C. Stein. Better online buffer management. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 199–208, 2007.
- [22] Z. Lotker and B. Patt-Shamir. Nearly optimal FIFO buffer management for DiffServ. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 134–142, 2002.
- [23] W. Willinger and V. Paxson. Where mathematics meets the Internet. *Notices of the American Mathematical Society*, pages 961–970, 1998.